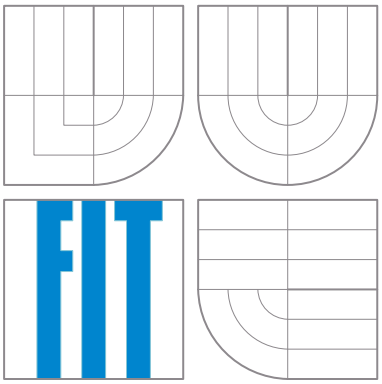


Dokumentácia projektu

# Interpret jazyka IFJ14

Varianta a/1/II

Rozšírenia MINUS, BASE, REPEAT, FUNEXP, ELSEIF, BOOLOP

14. decembra 2014


| | | |
|---|---|---|
| **Vedúci tímu 002:** | **Dávid Mikuš (xmikus15)** | 20% |
| Členovia: | Peter Hostačný (xhosta03) | 20% |
| | Tomáš Kello (xkello00) | 20% |
| | Adam Lučanský (xlucan01) | 20% |
| | Michaela Lukášová (xlukas09) | 20% |

# Obsah

# 1 Úvod

Cieľom projektu bolo vytvoriť funkčný interpret jazyka IFJ14, ktorý má veľmi podobnú syntax ako jazyk Pascal.

Táto dokumentácia pojednáva o základných princípoch použitých pri riešení.

# 2 Riešenie projektu

V tejto kapitole budú rozpísané jednotlivé časti s dôrazom na vlastné riešenie.

Vstupom interpretu je zdrojový súbor, ktorý je priamo vykonaný, namiesto generovania výstupného spustiteľného súboru.

## 2.1 Lexikálna analýza

Vstupný súbor reprezentovaný znakmi je nutné rozdeliť na „tokeny“. O túto činnosť sa stará tzv. „scanner“, ktorý je reprezentovaný konečným automatom (viz prílohu B). Dosiahnutie koncového stavu značí úspešný nález tokenu.

Každý načítaný token je pripojený na koniec vektoru. Z hľadiska interpretu sa najprv načítajú všetky tokeny, až potom sa s nimi ďalej pracuje v syntaktickej analýze, kvôli jednoduchšiemu spracovávaniu chýb. Ak by sa nachádzala lexikálna chyba na konci zdrojového súboru, mohlo by dôjsť k syntaktickej chybe skôr než by bolo detekovaná lexikálna.

Konverzie medzi reťazcami a číslami sa vykonávajú ihneď po vrátení platného tokenu. Konverzie v rámci escape sekvencie v reťazcovom literáli sa vykonávajú priamo v automate pri prechode medzi stavmi.

## 2.2 Syntaktická a sémantická analýza

Syntaktická analýza (parser) bola riešená zadaním cez rekurzívny zostup podľa gramatiky v prílohe C. Parser kontroluje pri svojej činnosti zároveň aj sémantiku (dátové typy), definície a deklarácie funkcií. Vyhodnocovanie výrazov zabezpečuje precedenčná tabuľka (viz. príloha D). Pre účely rozšírenia bol pridaný operátor unárne mínus.

Inštrukcie pre interpret sú generované okamžite na inštrukčnú pásku, bez dodatočného medzivýstupu.

## 2.3 Optimalizácie

Vzhľadom k možnosti sútaže o najrýchlejší interpret sme sa rozhodli zaviesť aj istú formu optimalizácií, a to hlavne výpočet hodnôt, ktoré sú známe ešte pri parsovaní, t.j. operácie s konštantnými hodnotami a mnohonásobné unárne mínus.

## 2.4 Inštrukcie a interpret

Ako najzaujímavejšia časť celého interpretu je generovanie a vykonávanie inštrukcií z inštrukčnej pásky vzhľadom k voľnosti riešenia tejto časti.

Interpret pri svojej činnosti pracuje so zásobníkom (o neobmedzenej veľkosti) a inštrukčnou páskou, ktorá obsahuje inštrukcie v trojadresnom kóde. Každý operand obsahuje informáciu o tom

- kde sa nachádzajú údaje (ďalej len lokalita):
  - globálne
  - lokálne (Base Pointer + Offset)
  - konštanta (okamžitá hodnota)
- akého sú dátového typu (integer, real, boolean, string)

Kedže interpret spracováva inštrukcie na úrovni C funkcií, pri každej inštrukcií sa najprv musí rozhodnúť akého je dátového typu a kde sa nachádzajú údaje pre každý operand, čo stojí veľké množstvo času pri samotnom behu v porovnaní so samotným výpočtom. Rozhodli sme sa teda vyriešiť tento problém elegantným spôsobom, a to zaťažiť parser o jednu činnosť navyše - generovanie konkrétnych inštrukcií pre interpret, obsahujúce v sebe zakódované lokality a dátové typy.

Vzhľadom k charakteru inštrukcií sme pomocou napísaného skriptu v jazyku Ruby vygenerovali 358 inštrukcií, ktoré priamo v názve obsahujú lokalitu a typy operandov. Inštrukcie sú uložené v poli ukazovateľov na funkcie (inštrukcie). Pri vykonávaní programu sa volá C funkcia pomocou ukazovateľa na danú funkciu ktorú v sebe obsahuje, čo zabezpečuje vysokú efektivitu výpočtu, kedy nie je nutné rozhodovať nad akými operandmi sa má daná činnosť vykonať. Jediné rozhodovanie v inštrukcií je pri testovaní inicializovanosti operandov a pri delení nulou. Celkovo bolo použitých 15 základných inštrukcií (`CALL`, `RET`, `MOV` ...) a 15 aritmetických (nerozvetvených na jednotlivé dátové typy a lokality).

Interpret beží v nekonečnom cykle, pričom končí v momente keď narazí na inštrukciu `HALT`.

### 2.5 Prekladový systém

Pre vlastnú potrebu projektu sme sa rozhodli využiť modulárny preklad z dôvodu testov pre jednotlivé moduly, kedy sa všetky objektové súbory (.o) okrem modulu obsahujúceho funkciu `main()` zabalia do archívu (.a) utilitou `ar`. Pre unit testy, ako aj spustiteľný interpret obsahujúci `main()` sa nalinkuje archív, ktorý obsahuje potrebné moduly.

## 3 Algoritmy a dátové štruktúry

### 3.1 Quicksort

Funkcia na zoradenie prvkov v poli. Ako prvé si zadefinujeme indexy okrajových hodnôt medzi ktorými sa nachádzajú prvky na zoradenie spolu s indexom prostrednej hodnoty ktorú určíme ako pivot.

Následne posúvame indexy z pravej a ľavej strany smerom k pivotu, v prípade že nevyhovujú postupnosti z ľavej strany od najnižšej hodnoty, vtedy tieto dve okrajové hodnoty prehodíme, až kým neprejdeme celým polom. Takýmto spôsobom presne definujeme pozíciu pivota v poli. Všetky prvky na ľavej strane od neho sú menšie a na pravej väčšie ako jeho hodnota, avšak nie sú v správnom poradí. Preto sa rekurzívne volá rovnaká funkcia, avšak už len nad pravou a ľavou stranou ktoré sú menšie. Takýmto spôsobom prejdeme celé pole ktoré sa stane usporiadaným.

Algoritmus je založený na mechanizme rozdeľovania (partition), ktorý preskupí prvky pola do dvoch častí tak, že v ľavej časti sú všetky prvky menšie (alebo rovné) istej hodnote a v pravej časti sú všetky prvky väčšie než táto hodnota [1].

## 3.2 Knuth-Morris-Pratt

Vyhľadávanie podreťazca v reťazci je vo vstavanej funkcií riešené algoritmom Knuth-Morris-Pratt, ktorý je založený na vytvorení masky hľadaného textu. Maska určuje programu ako sa má chovať v prípade, že sa písmeno hľadaného textu nezhoduje.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| A | T | T | A | T | A | C | A |
| 0 | 0 | 0 | 1 | 2 | 1 | 0 | 2 |

Tabuľka 1: Príklad masky algoritmu KMP

Maska je pole o dĺžke hľadaného textu a obsahuje celočíselné nezáporne čísla. Ku každému písmenu je priradené číslo ktoré udáva index, kam sa má program vrátiť v prípade, že nenastane zhoda na danej pozícií. Príklad takej masky pre slovo „ATTATACA“ sa nachádza v tabuľke 1.

Význam tohto algoritmu vidíme v prípade, že uprostred slova sa nachádza rovnaká sekvencia ako na začiatku prehľadávaného textu. To znamená, že ak vo vstupnom texte nastane nezhoda na indexe 6, avšak predchádzajúcich 5 znakov už bolo skontrolovaných, podľa masky vieme vyčítať posun na index 2. Inak povedané, písmena s indexmi 4..5 sú zhodné s 1..2 a preto ich netreba kontrolovať znovu, ale môžeme pokračovať na indexe 3. Ak ani na treťom indexe neuspelo porovnanie, vraciame sa na začiatok, t.j. index „0“. Takto pokračujeme až kým nedosiahneme koniec vstupu alebo nenájdeme celý reťazec vo vstupnom texte.

## 3.3 Tabuľka s rozptýlenými položkami

Táto dátová štruktúra bola použitá na uschovávanie lokálnych symbolov v danej funkcií. Jej výhodou je rýchlosť hľadania symbolov. Základom celej tabuľky je pole ukazovateľov na jednosmerne viazané zoznamy. Každá položka v zozname obsahuje ukazovateľ na následníka a samotný symbol.

Na základe symbolu sa pomocou tzv. *hashovacej* funkcie, ktorá mapuje vstupný reťazec na index získa ukazovateľ na prvú položku v zozname, od ktorej sa sekvenčne prechádza až dokým sa nenájde požadovaná položka, alebo nie je dosiahnutý koniec zoznamu.

## 3.4 Vector - pole s neobmedzenou kapacitou

Pre účely interpretu bolo nevyhnutné vytvoriť vlastný dátový typ pre prácu s polami rôznych dĺžok, ktoré bežné polia neumožňujú, definovali sme si typ `Vector`, inšpirovaný vektorom z jazyka C++.

`Vector` je štruktúra, ktorá si nesie informácie o alokovanom a použitom počte položiek spolu so samotným ukazovateľom na začiatok alokovaného priestoru. Pre prácu s vektorom sú definované špeciálne funkcie, ktoré v prípade zaplnenia realokujú potrebný priestor na zápis položky. Realokuje sa vždy na dvojnásobok predchádzajúcej veľkosti.

## 3.5 Reťazec

Podobne ako dátový typ `Vector` bolo nutné vytvoriť aj `String`, ktorý funguje na prakticky tom istom základe realokácie v prípade potreby, avšak s odlišnými operáciami (konkatenácia, dĺžka).

# 4 Práca v tíme

Vývoj na projekte sa započal dňa 25. augusta prvým commitom v Git repozitári. Pred samotným zadaním projektu sa započala práca na na prekladovom systéme a hneď potom na základných štruktúrach ako napr. `Vector` a `String`. Počas riešenia projektu boli 3 schôdze celého tímu, kde sa stanovil plán na nadchádzajúce týždne, pričom prvá schôdza bola 30.9.2014. Rozsah projektu tvoril isté problémy s načasovaním schôdzí, na ktorých bolo nutné aby sa jej zúčastnili všetci členovia. Počas samotného vývoja činila problém časová náročnosť projektu a náväznosť na prednášky, tak sme sa rozhodli dopredu naštudovať problematiku, aby sme mohli posledný týždeň testovať a odstraňovať chyby.

## 4.1 Rozdelenie práce

| | |
|---|---|
| Dávid Mikuš | Vedúci tímu, parser, testovanie |
| Peter Hostačný | Vyhodnocovanie výrazov, testovanie |
| Tomáš Kello | Algoritmy, pomocné štruktúry, dokumentácia |
| Adam Lučanský | Scanner, interpret, dokumentácia |
| Michaela Lukášová | Scanner, dokumentácia |

# 5 Záver

Projekt sa nám podarilo úspešne dokončiť a otestovať v požadovanom čase s optimalizáciami a mnohými rozšíreniami. Projekt bol náročný a mnohokrát išlo „do tuhého“. Berieme to ako veľkú skúsenosť do budúcna, hlavne prácu v tíme.

# Literatúra

[1] HONZÍK, Jan. *Algoritmy – studijní opora*. Brno: Vysoké učení technické, 2014.

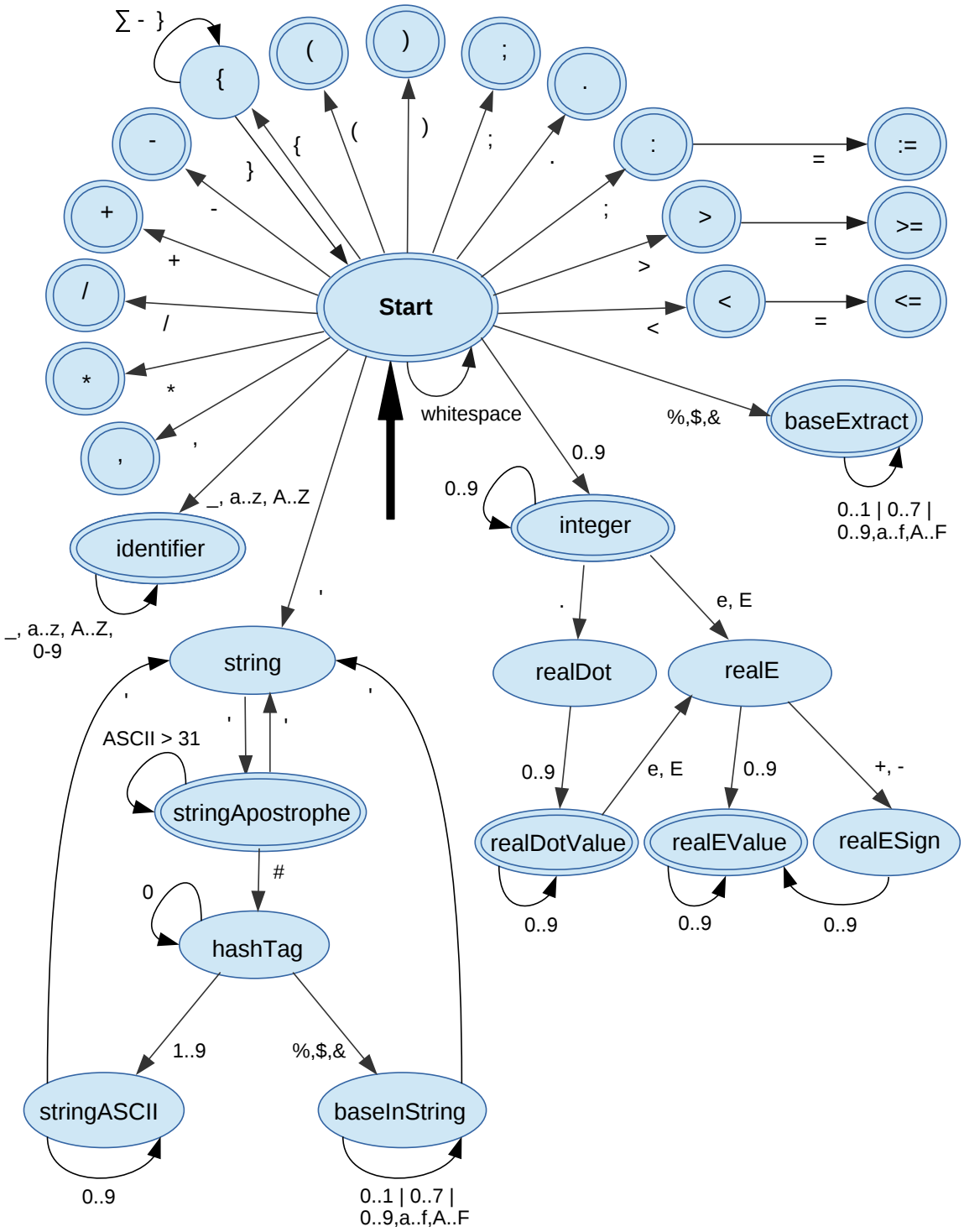
# A Metriky kódu

**Počet riadkov zdrojového textu:** 11429 riadkov

**Veľkosť statických dát:** 456B

**Veľkosť spustiteľného programu:** 149.8KiB

# B Konečný automat lexikálneho analyzátora



# C LL Gramatika

**1** : *PROGRAM* → *VAR_DECLR FUNC COMPOUND_STMT* .
**2** : *VAR_DECLR* → var *VAR_DEF*
**3** : *VAR_DECLR* → $\epsilon$
**4** : *VAR_DEF* → id : type ; *VAR_DEFN*
**5** : *VAR_DEFN* → id : type ; *VAR_DEFN*
**6** : *VAR_DEFN* → $\epsilon$

**7** : *FUNC* → function id *PARAM_DEF_LIST* : type ; *FORWARD FUNC*
**8** : *FUNC* → $\epsilon$
**9** : *FORWARD* → forward ;
**10** : *FORWARD* → *VAR_DECLR COMPOUND_SEMICOLON_STMT*

**11** : *PARAM_DEF_LIST* → ( *PARAMS_DEF* )
**12** : *PARAMS_DEF* → id : type *PARAMS_DEF_N*
**13** : *PARAMS_DEF* → $\epsilon$
**14** : *PARAMS_DEF_N* → ; id : type *PARAMS_DEF_N*
**15** : *PARAMS_DEF_N* → $\epsilon$

**16** : *TERM_LIST* → ( *TERMS* )
**17** : *TERMS* → term *TERMS_N*
**18** : *TERMS_N* → , term *TERMS_N*
**19** : *TERMS_N* → $\epsilon$

**20** : *COMPOUND_STMT* → begin *STMT_E* end
**21** : *COMPOUND_SEMICOLON_STMT* → *COMPOUND_STMT* ;
**22** : *STMT_LIST* → $\epsilon$
**23** : *STMT_LIST* → ; *STMT STMT_LIST*
**24** : *STMT_E* → *STMT STMT_LIST*
**25** : *STMT_E* → $\epsilon$
**26** : *STMT* → id := *EXPR*
**27** : *STMT* → if *EXPR* then *COMPOUND_STMT IF_N*
**28** : *STMT* → while *EXPR* do *COMPOUND_STMT*
**29** : *STMT* → repeat *STMT STMT_LIST* until *EXPR*
**30** : *STMT* → *COMPOUND_STMT*
**31** : *STMT* → readln ( id )
**32** : *STMT* → write *TERM_LIST*

**33** : *IF_N* → else *COMPOUND_STMT*
**34** : *IF_N* → $\epsilon$

# D Precedenčná tabuľka

| | -[1] | not | * | / | and | + | - | or | xor | < | > | <= | >= | = | <> | ( | ) | f | , | $ | var |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -[1] | $H$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| not | $R$ | $H$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| * | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| / | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| and | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| + | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| - | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| or | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| xor | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| < | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| > | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| <= | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| >= | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| = | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| <> | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $S$ | $R$ | $S$ | $R$ | $R$ | $S$ |
| ( | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $H$ | $S$ | $H$ | $E$ | $S$ |
| ) | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $E$ | $R$ | $E$ | $R$ | $R$ | $E$ |
| f | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $E$ | $H$ | $E$ | $E$ | $E$ | $E$ | $E$ |
| , | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $H$ | $S$ | $H$ | $E$ | $S$ |
| $ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $S$ | $E$ | $S$ | $E$ | $E$ | $S$ |
| var | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $R$ | $E$ | $R$ | $E$ | $R$ | $R$ | $E$ |

R – reduce, H – handle, S – shift, E – error

[1] Unárne mínus